

CONNECTING *the* D *to the* DESIGN *of* Wo

WHEN WE DESIGN INTERACTIVE SOFTWARE WE ARE ALSO DEFINING MUCH ABOUT THE WORK of its users. The software embodies a model of work processes for its end users because part of its job is to manage the content, format, and sequencing of the information that users need to do their work. The effect is that any application will preferentially enable certain work processes, and users will have to work harder to follow any others. It is actually unavoidable. Developers may try to avoid the responsibility for promoting a particular work model by oversupplying information or features for flexibility. But this strategy is futile. It simply loads the user with an additional process, one for dealing with the resulting clutter. The effect is the same whether the application was developed within a particular organization or purchased off the shelf.

Carroll and Campbell [2] and Suchman [11] have also argued that software embodies user work processes, but stopped short of examining the profound implications for organizations and businesses. They need to respond to accelerating change in the world by improving their work processes continuously, and in some cases by reengineering them completely. Often an information system will be the principal means to implement the desired new process. However, applications that don't carefully build in the desired work process actually end up selecting one by implication rather than by design.

The implied work model may actually contradict a good business process for performing the work. Then the power of computing can turn into a kind of ubiquitous, electronic superbureaucrat that imposes unnecessary overhead while stubbornly frustrating users' attempts to work productively.

Designing Work versus Designing Software

In order to design work deliberately we need to understand the difference between work design and software design. The design of work and the design of software are distinctly different domains. The two designs must interact, but they must each also function in their respective local environments, which are fundamentally different. The analysis and design of work deals with activity that will be executed in the physical world, where its goal is to achieve physical qualities such as better resource utilization, reduced cycle time and variance, reduced costs and waste, or to generate products with real-world qualities. In contrast, software design has historically emphasized completeness and consistency, and the efficiency of code that will be executed electronically.

KEITH A. BUTLER, CHRIS ESPOSITO, AND RON HEBRON

DESIGN *of* SOFTWARE

RK *When the software's preference contradicts good user work practice the power of computing quickly becomes an electronic bureaucrat—frustrating productivity while imposing overhead.*

A representation language for designing work should explicitly account for the utilization of the physical resources and information that are needed to accomplish the work. It should also support identification of performance obstacles in an existing work process, and the analysis of how to improve it. In order to meet these requirements, a representation language for modeling work should also provide some metric of goodness, so we can predict when a work design will actually make an improvement, or compare design alternatives.

Work design and software design require two different types of languages to model two different worlds that are understood by different people. Kyng [5] argued persuasively that the logical responsibility for work design belongs to the user community, while application development is the domain of computing technologists. This division of labor seems like a realistic view. Users have the most intimate understanding of their work. But technology changes the possibilities for accomplishing work, and users cannot be expected to know enough about it. It is the job of computing professionals to know about new technical possibilities and, as importantly, the limitations on the technology that is available.

Integrating the two views can be difficult, expensive and unreliable. But it is also essential because design involves making rational trade-offs between the costs of technology and the benefits of its impacts. In this context we need to understand how the benefits to the work process trade off against the cost,

timeliness, or technical feasibility of the new system.

Iterating back and forth between the two types of design seems like a practical solution. But there are a wide variety of social and economic factors that have historically kept developers and users from collaborating more effectively or more often [5, 8]. The one that we will focus on here is overcoming the technical difference between designing work and designing software, and the obstacles it poses to iterative design.

Making the Connection

Attempts to make a closer connection between the design of software and the design of work have begun evolving rapidly. In the software modeling community the Unified Modeling Lan-

guage (UML)¹ for the analysis and design of object-oriented software (OOAD) has recently added user-centered views such as use-cases and user activity diagrams. These are important developments from the standpoint of escalating the importance of setting and satisfying user requirements. They also provide a much more systematic connection from user requirements to all the aspects of implementation. However, use-cases do not currently satisfy our requirements for a language to support work design. They are an effective way to document work processes but do not support prob-

¹More detailed information on UML is available at www.rational.com/uml/

lem identification, improvement analysis, or any metric of goodness.

The methodology community has also made important contributions. For example, Kyng [5] described a thorough method for integrating work design with the design of supporting software. His primary representations of user work are scenarios. To represent the system Kyng applies user interface prototypes.

We are attempting to extend the work of researchers like Kyng and also of UML in several directions. One important limitation of Kyng's method is the scope. It's main focus is on user interface design. But we believe that during the early stages of design the entire architecture of a proposed system must be analyzed for cost and feasibility, so the representation of the proposed system should not be limited to the user interface or its neighboring components. UML does offer the advantage of

ers. Also, interviewing and modeling are valuable opportunities to observe nuances and obstacles to good work performance or to note valuable design ideas for later use. A work model can also identify user data needs and map them to user tasks, regardless of implementation details.

We also assert that a business process is actually some aggregation of user tasks. In order to improve the quality of business processes and products with interactive computing, we must be able to make a connection between the way people work, both singly and in combination, with improved performance at the level of a business process.

We focus our modeling to capture the conceptual data model along with the logical work process and rules that must persist, regardless of the information system being used. If the model is open with respect to any particular implementation, it can produce insights about alternate ways in which technology

{ Building the work model can expose important gaps
in our understanding about the way users perform
work, both singly and in cooperation with others. }

connecting its use-cases to the design of the entire system architecture via event sequence diagrams. However, use-cases and other scenarios that rely on free text as their representation language seem inherently limited in their ability to meet the requirements for work design that we explained in the previous section.

In this article we will look at two emerging standards: UML for modeling the functional aspects of software; and IDEF3 for modeling work activity [6]. We will give simple examples to illustrate how IDEF3 can be used to design work, and then show how important aspects of a UML software model can be derived from the IDEF3 model. We will also show how the ability to relate the two enables a practical, iterative method for the design of work and the design of supporting software.

User-Centered Models of Work

There are strong reasons for building a model of work to represent the users' view. It makes our assumptions about user work explicit. The activity of building the work model can expose important gaps in our understanding about the way users perform work, both singly and in cooperation with oth-

ers. This point is also important because it frees us from the necessity of modeling how the users perform current work in too much detail. Analysis actually has two objectives: The first is to understand the current situation; the second is to produce requirements for improvement.

A work process model also calls for understanding who the users will be, their goals, roles and tasks, and the way they think about their work. Defining the intended user population, both in terms of professional and computing skills, is essential in determining who will be at the center of the design process. Representative samples of users are needed to gather knowledge about their application domains and to support user testing. Often the user population is heterogeneous, in which case subgroups and their relative proportions also need to be defined.

Modern methods for work modeling began when industrial engineers needed to plan and prescribe activity as a sequence of steps in order to perform a physical task [7]. Precedence diagrams and hierarchical decomposition became popular techniques. Precedence represents the temporal sequence or logical dependencies in which activity units take place.

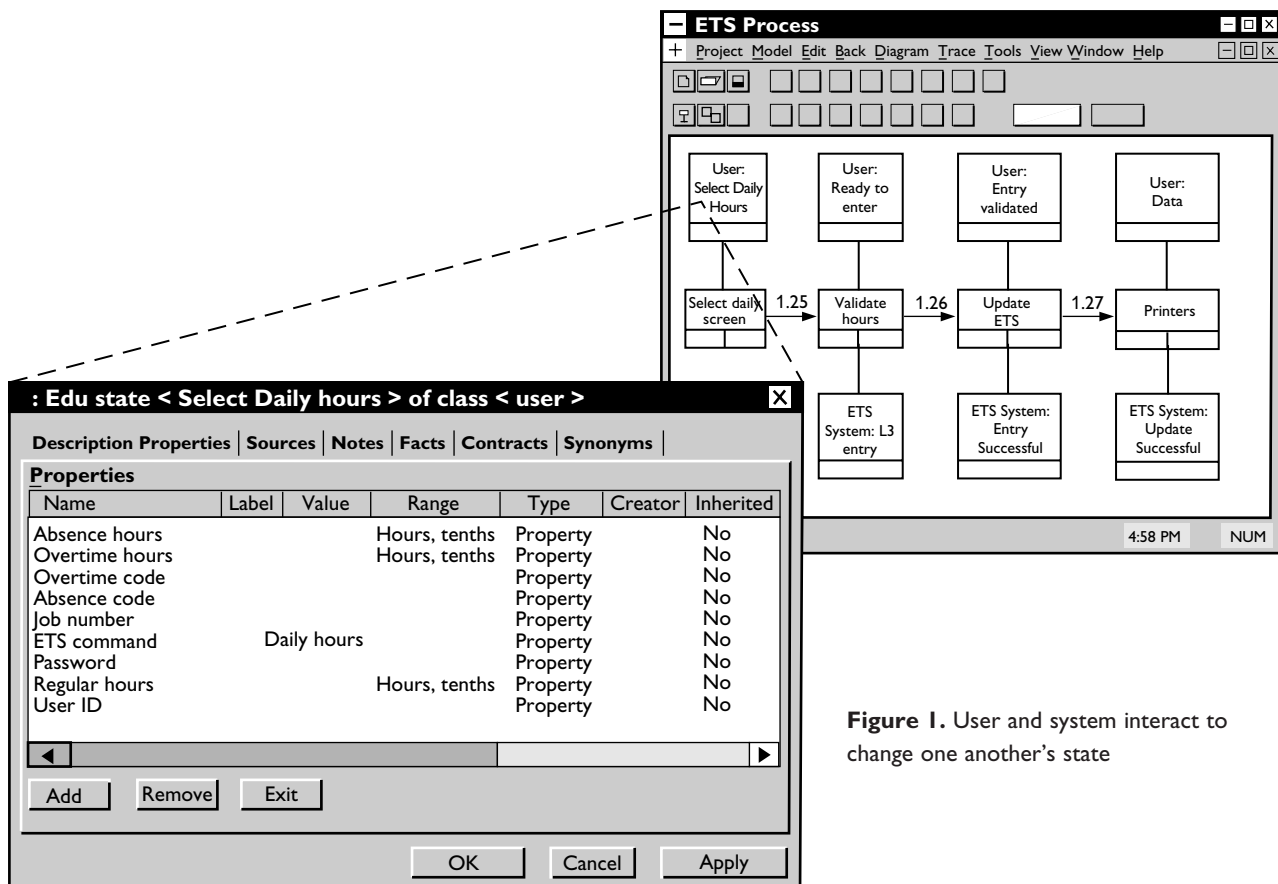


Figure 1. User and system interact to change one another's state

Hierarchical decomposition is a common technique for managing large or complex processes in which sub-processes can be nested within higher-level processes.

One of the best specified languages for representing work processes in the physical world is IDEF3, developed by the U.S. Air Force to standardize a technique for stating requirements [6]. Unlike many ad hoc methods for task analysis, IDEF3 has a well-defined ontology for representing the processes that accomplish work. In the specifications of the Workflow Management Coalition this type of model corresponds in part to a process definition tool [4]. But IDEF3 has a much richer representation than the flow of work. It explicitly represents the people, machines, and other resources that must participate to execute the processes. The behavior characteristics of these participants, in turn, support discrete event simulations to evaluate how well a process will perform.

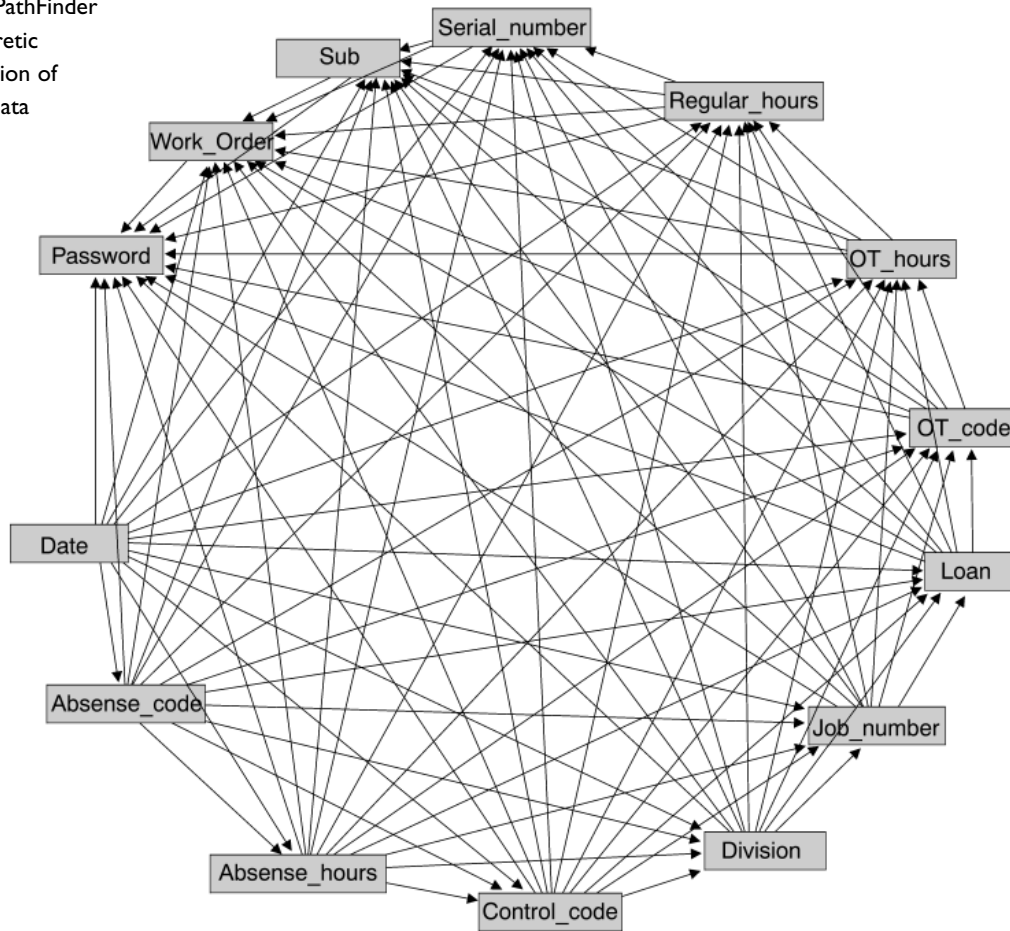
When we apply IDEF3 for the design of information work, the lowest event level is a user task, in which users interact with information systems to accomplish work goals. Figure 1 gives an example of this level. The work domain for this illustration is employee time-keeping. It shows the user process for employee time-keeping with an existing IMS data-

base application called ETS. We deliberately chose a simple case with one user and one information system to illustrate the features of the modeling language. The same language could also represent multiple users working in collaboration and supported by many information systems. The screen shots have been assembled from an unreleased tool for enterprise-modeling that has integrated several different IDEF views [12].

In Figure 1 the upper screen shows how the user and the ETS system interact as participants in a series of process steps to change one another's states until the user's goal state *Done* is reached. Each of the participants maintains its identity although the different steps may require it to provide different information and be in a different state.

The lower screen shows how to use the property editor for a participant to represent the information that it must have in order to proceed to each step of a task. The property values, in turn, determine the state of a participant. In our example the *user* must be in the *Select Daily Hours* state in order to participate in the step for *Select Daily Screen*. The state of *Select Daily Hours* means that the *user* possesses values for the needed information properties. Specifically, *user* has a value *ETS command*, while all the

Figure 2. Pathfinder
graph-theoretic
representation of
proximity data



other *user* properties are null, meaning they are not required for this step.

If the information type has not appeared before, we add it as a property to the participant that serves as its source. Wherever an information type is used in a task, we assign it the needed value in its containing participant. Wherever the information type is not needed in a task the property value will be set back to null. This technique allows us to incrementally capture the information support for an existing work process. In our example when the model is complete the *user* will contain all the information types it needs to participate in all time-keeping tasks. Alternately, when we are designing an improved work process the same technique allows us to define information support requirements.

In this way we can exploit participant states to capture the information required to support a given work model. It allows us to map specific information types to specific steps in a work process. This feature allows us to calculate the way each information type must be distributed to support the work process. As will be shown, this is a key principle for connecting the work model to software design.

Translating Information Distributions into Object Classes

One of the defining features of a good, high-level object is that it should contain information attributes that need to be used in conjunction with one another. These attribute clusters can be derived from our IDEF3 model by analyzing the utilization of information over the tasks. We use a graph-theoretic approach called Pathfinder to analyze utilization patterns and discover clusters of information types. A complete description of the techniques, algorithms and mathematics involved can be found in [1, 3, 10].

Pathfinder is a data reduction technique that has proved quite useful for discovering clustering, and has been successfully used elsewhere for tasks such as eliciting and representing domain expert knowledge structures [10]. It is similar to, but more general than, its better-known sister, hierarchical cluster analysis. This section describes the basic steps for reducing and analyzing the raw co-occurrence data and deriving definitions for candidate business objects.

One of the intuitions that drives our approach is the idea that a pair of data elements with identical

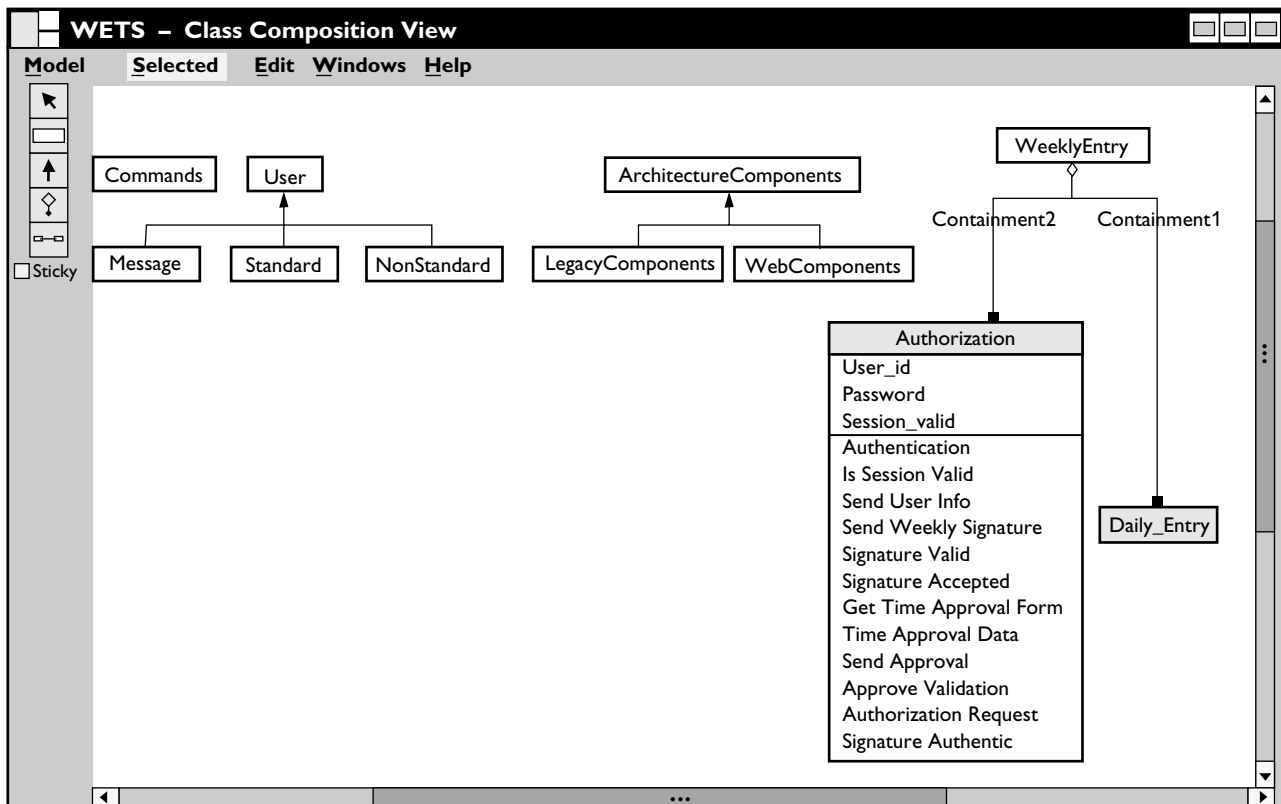


Figure 3. Class diagram for designing software

usage distributions is more likely to wind up as part of the same object than a pair of data elements that are never used together in any task. The Jaccard coefficient is an equation for calculating the similarity in usage vectors across tasks for each pair of information types. From these coefficients we calculate a matrix of proximity values so that each value represents some measure of distance between information types. The result is that information types with similar usage vectors are 'near' one another. This matrix of proximities can be displayed as a complete network, that is, a graph with an edge between every pair of nodes (information types) and with the proximity value as the numerical weight attached to the edge.

Figure 2 shows the results of a graph-theoretic representation. It displays a network structure of arcs for the strength of associations among all pairs of information types used in employee time-keeping. The positioning of the nodes in Figure 2 is only an aid to understanding. Unlike multidimensional scaling, the spatial relations are not usually significant in themselves. The structure of the network is of greatest interest. A circular layout is shown in Figure 2. Two other layouts, orthogonal and symmetric, are also available.

Pathfinder removes the weakest associations between information types so that the resulting net-

work only has edges where there is a strong connection between elements joined by these edges. As an aid to understanding these networks, they are displayed in graphical rather than tabular form. The results of our example Pathfinder cluster analysis are shown in Figure 2.

A variety of graph-theoretic structural and cluster analyses are available to further reduce the data and derive definitions of candidate business objects. Studies have shown that interesting clusters appear in the structure as cliques, near-cliques and stars [3]. Algorithms have been implemented for identifying these structures, and also for groups whose members are perfectly correlated. All of these patterns are subgraphs that can be identified using the cluster-finding algorithms. Once clusters have been identified, a node editor is used for placing the members of a cluster into a higher-level node. The resulting cluster definitions can be exported to a tool for OOAD for feasibility analysis and design of the supporting software.

Technology-Centered Software Modeling

In the early 1990s Rumbaugh and his colleagues [9] developed an excellent and well-written manual for the practice of OOAD. There is a companion article by Mylopoulos, Chung, and Yu in this section that explains OOAD and traces its history. In recent years

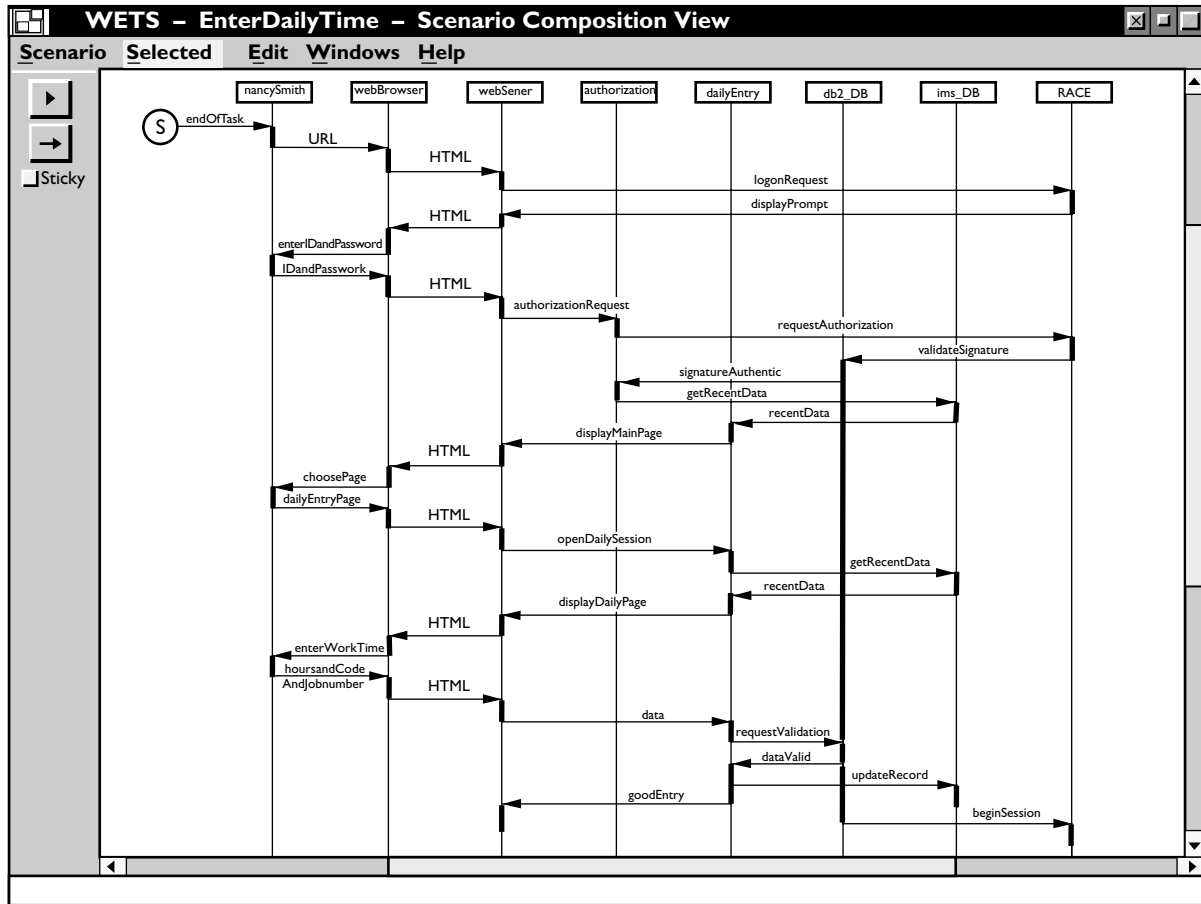


Figure 4. UML event sequence diagram

the modeling languages, methods and tools for OOAD have matured rapidly. In an important industry-wide development the Object Management Group (OMG) has defined a clear notation for OOAD in its UML. OMG recognized that no single view of software was sufficient to capture its complexity, so UML integrates such popular views as class diagrams, use-cases, and event sequence diagrams.

Figure 3 shows the Authorization class, whose definition was downloaded directly from cluster analysis. The tool for OOAD is an alpha version of IBM's OBJChart. The clusters of information types that were derived from usage patterns have now been loaded as *classes* into a tool for software analysis and design.

The class for Authorization contains the information types that need to be used together in the work process as *attributes*. The object class for Authorization has both meaning and match for the work process it is intended to support.

Figure 4 shows the event sequence diagram developed by the software architect. Event sequence diagrams have also been called *object interaction*

diagrams. They are a useful way to analyze how the objects of a system must interact in order to satisfy a use-case, a scenario, or a user work process. In the event sequence notation, a column represents the life cycle for each object from top to bottom, and the horizontal lines represent the flow of responsibility. By making this flow explicit an analyst or simulation can determine whether the flow is complete and whether each object can satisfy the requirements of its responsibilities. Figure 4 shows the event sequence diagram for our employee time-keeping example.

Figure 4 represents the objects and events that must take place for software to support the time-keeping work process that was designed earlier. However, there is a different focus. The design problem being attacked is technology-centered, that is, how to make software systems function in the required way. In order to perform this type of analysis a software architect would have to add technical objects, such as the IMS database and the RACF security system, in order to assess the technical feasibility of supporting the work process as it was defined.

Converging the Design of Work and the Design of Supporting Software

Our intent is to support a technical engineering dialog between work process (WP) design and OO design, based on quantifiable value to the business and technical feasibility for the software. The goal of this interactive design flow is a well-matched pair of designs: a WP design that is valuable and *Implementable*; and a software design for business components that is feasible and cost effective.

Figure 5 shows how such collaboration can take place to promote convergence of the two design tracks. The needed life cycle begins with an analysis of an existing work process and proceeds through component reuse for continuing work improvements. The method is similar to that advocated by Kyng [5] but it incorporates more of the software developer's responsibilities. It also clearly states what each iteration should try to accomplish.

Initially the *As-is* WP should be modeled in terms of user tasks and data to understand what improvements are needed and how they may be accomplished. The WP's performance qualities can be measured or estimated to establish a baseline or to help diagnose the sources of any poor performance.

The *As-is* software should also be modeled. Event sequence diagrams can document how software currently works. The *As-is* software constrains the *As-is* WP. That may be good or bad for the WP, but currently it is not dependably good.

The work analyst then determines how user work processes should be changed. The new user work processes are documented in the context of a *Desired* WP model. The *Desired* model should generate improved performance estimates on quality metrics. The work model can be queried to obtain usage distributions on data over all user tasks. The distributions help define data requirements for supporting the *Desired* WP. As was shown earlier, they can provide candidate definitions for business objects that are given to the software designer as classes for OOAD. Since containment defines the source of the information, this part of the model is also valuable for identifying reusable portions of legacy systems. By explicitly modeling data sources we can also begin defining reuse potential for any legacy systems that may be available.

The software architect takes the preliminary business requirements in the form of classes to develop a *Corresponding* model of software. The classes and use-cases from the WP can be elaborated into interaction diagrams to analyze the technical feasibility of the required functionality. The architect may compare requirements to common

business objects or other libraries for reuse.

The results of the feasibility analysis on the *Corresponding* object model can provide feedback to the work designer as to whether the software can be created to implement the *Desired* WP. It could well be that the technical feasibility analysis by the OOAD indicates a level of technical risk that is too high. Alternately, technical possibilities may enable greater improvements than the business analysts originally realized. This prospect is just one reason why the designers for work and the designers for software must collaborate while they iterate their respective designs.

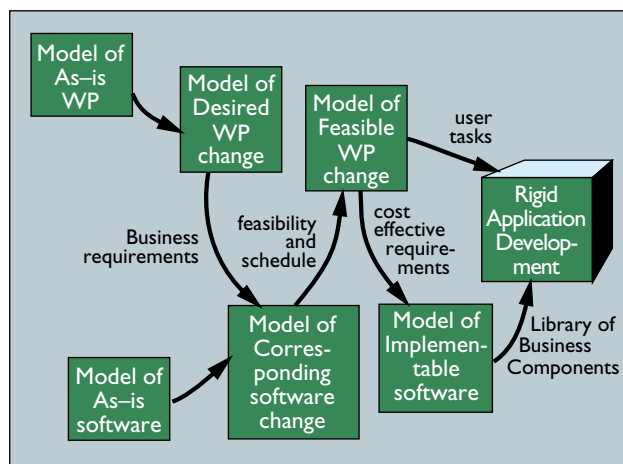


Figure 5. Connections in the converging life cycle

The work designer can then perform a cost-benefit analysis and a sensitivity analysis to determine priorities for *Feasible* changes. This decision will likely require several iterations between WP and software models because the software analysis can reveal both technical constraints and technical possibilities. They need to be analyzed for their impact on the WP. Several alternative combinations of WP and supporting software may be examined in order to arrive at a well-matched pair of effective designs.

Further Technology Development

The methodological steps we have presented involve the development of the WP model, including data requirements, from use-cases, and the derivation of business component definitions from cluster analysis routines. The definitions can then be loaded as classes for further OOAD work. But there are other types of models that will be needed to enable a Converging Life Cycle.

Business value simulation. There are two important, related limitations on use-cases and scenarios.

Neither has any metric of “goodness” nor any methodical way to estimate added value. When a designer is faced with a choice between two alternatives for a use-case there is no analytical way to decide which is better. Estimating a return on investment is one of the most common ways to make business decisions. IDEF3 is complete enough to support discrete event simulations of the *Desired WP*. We have begun simulating work processes to estimate the added value of a new work design. A process-oriented view of use-cases can be built in UML as Activity Diagrams. We should also investigate how Activity Diagrams could be extended to support the needed analysis.

Software project estimation. Another missing connection in the Converging Life Cycle (Figure 5) is a cost and schedule estimate for developing the needed support software. In our experience, UML seems useful for assessing technical feasibility, but by itself feasibility does not provide cost information. If a UML model could provide input to a tool for estimating the cost and schedule for the object-oriented software development, then an investment decision could be supported. The costs and time to support the new work process could be compared to the benefits estimate for implementing it. The structure of UML is elaborate and it seems like an ideal situation for investigating how well it could support project estimation.

Reuse estimation. The connection between work models and object models may have additional benefits for estimating the reusability of software components. The PathFinder networks provide a methodical means to estimate the reuse potential for business objects to respond to changing business needs. From a statistical point of view, this is equivalent to asking how sensitive the design of the underlying software system is to changes in the work processes that it supports.

Lessons Learned

The tool suite we have described is an advanced prototype and only one example of how software and work design can be connected through iterations. The advantages of our particular example are three-fold: it is a reliable and replicable method; it addresses the entire system architecture; and once the first pair of models has been built we can move through additional iterations quickly.

The benefits of establishing the connection between work and software design are much more profound. If work process improvement is the goal, then often an information system is the means to achieve it. When the goal and its implementation

are defined in isolation the predictable result will be partial success at best. Our tools are not a substitute for insight or creativity in design, but they do promote a focus on the key issue of adding value to work via computing applications that are technically feasible and cost effective.

By planning a project around the two converging design tracks we make an explicit place for both, and at the same time promote an understanding of the roles that the interdisciplinary project team should perform. In our experience the results are very promising in terms of excellent customer satisfaction, on-time and within-budget delivery of high-quality software, and very high team morale. **G**

REFERENCES

1. Butler, K., Esposito, C., and Klawitter, D. Designing more deeper. In *Proceedings of DIS'97*. ACM, 1997.
2. Carroll, J.M. and Campbell, R.L. Artifacts as psychological theories: The case of human-computer interaction. *Behaviour and Info. Tech.* 8, 4 (1989), 247–256.
3. Esposito, C. A graph-theoretic approach to conceptual clustering. In Schvaneveldt, R.W. *Pathfinder Associative Networks: Studies in Knowledge Organization*. Ablex, Norwood, NJ.
4. Hollingsworth, D. The workflow management coalition specification. *Workflow Management Coalition Document No. TC00-1003*, 1995.
5. Kyng, M. Making representations work. *Commun. ACM* 38, 9 (Sept. 1995), 46–55.
6. Mayer, R., Cullinane, T., deWitte, P., Perakath, B., and Wells, M. *Information Integration for Concurrent Engineering (IICE): The IDEF3 Process Description Capture Method Report*. USAF Armstrong Lab, 1992.
7. McCormick, E.J. *Job Analysis: Methods and Applications*. AMACOM, NY, 1979.
8. Poltrock, S.E. and Grudin, J. Organizational obstacles to interface design and development: Two participant-observer studies. *ACM Trans. Computer Human Interaction* 1, (1994), 52–80.
9. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, NJ, 1991.
10. Schvaneveldt, R.W. *Pathfinder Associative Networks: Studies in Knowledge Organization*. Ablex, Norwood, NJ, 1990.
11. Suchman, L. Representations of work. *Commun. ACM* 38, 9 (Sept. 1995), 33–34.
12. Tissot, R. and Crump, W. An integrated enterprise modeling environment. *International Handbook of Architectures of Information Systems, Volume 1*. Springer-Verlag, 1998.

KEITH A. BUTLER (keith.a.butler@boeing.com) is an Associate Technical Fellow in the Applied Technology Division at Boeing.
CHRIS ESPOSITO (christopher.esposito@boeing.com) is an Associate Technical Fellow in the Applied Research and Technology Division at Boeing.
RON HEBRON (ron.hebron@boeing.com) is a senior system modeling analyst in the Applied Research and Technology Division at Boeing.

Copyright is held by the Boeing Company.
